LIGHTWEIGHT GLOBAL DISTRIBUTION MECHANISM

Technical Field

The present invention relates generally to data processing systems and more particularly to a lightweight global distribution mechanism.

Background of the Invention

In a distributed system there is often a need to distribute packages to various computer systems within the distributed system. These packages may contain programs, patches, documents or other files. In conventional systems, distribution requires the computer systems to register with a master to identify what packages are to be distributed to the respective computer systems. The master then issues a publishing event indicating that a package is available, and the package is distributed to the parties that registered for the package.

Unfortunately, this conventional approach suffers from some drawbacks. For example, there is a great deal of overhead incurred in providing support for the registration. A registry must be maintained, and resources must be provided to enable the computer systems to register with the master. In addition, there is no mechanism for a computer system to anonymously install packages as needed.

Summary of the Invention

The present invention addresses the limitations of conventional systems by providing a lightweight global distribution mechanism for distributing packages. The present invention provides a distribution mechanism that does not rely upon registration, and thus, does not incur the overhead associated with maintaining a registry. In addition, the present invention allows clients to anonymously download or install the packages. The present invention is platform independent such that it may be practiced on a number of different types of platforms.

In accordance with one aspect of the present invention, a method is practiced in a distributed system that includes a publishing master and a client system. The publishing master is provided with an index of available packages for loading. The packages may take many forms, including the form of a patch, a document or a hypertext markup language (HTML) file, for example. The index is accessed on behalf of the client system to identify at least one selected package for installation at the client system. The selected package is then loaded on the client system. The selected package may be loaded without registration of the client system with the publishing master, and the publishing master may be unaware of the loading of the selected package at the client system.

In accordance with a further aspect of the present invention information that identifies packages that may be installed at a client is stored at a server. Data that identifies packages to which the client subscribes is stored at the client. The information identifying the packages that may be installed at the client and the data that identifies packages to which the client subscribes are used to determine which of the packages are to be installed at the client.

In accordance with a further aspect of the present invention, a method is practiced in a computer network that has a server and a client. In accordance with this method, data is provided at the server regarding what packages are available. An itemization of packages to which the client subscribes is provided at the client. The following steps are then repeated multiple times: data is accessed regarding what packages are available, and data is accessed regarding the itemization of packages to which the client subscribes to determine what selected packages to load and subsequently the selected packages are loaded at the client. The steps that are repeated multiple times may be repeated at periodic intervals or upon demand by a user.

In accordance with an additional aspect of the present invention, a distributed system comprises a publishing master that holds an index of packages that are available for installation and a repository for storing the packages that are available for installation. The distributed system also includes a client computer system for identifying selected packages among the packages indexed by the index of the packages. The client computer system obtains the selected packages from the repository and installs the selected packages.

In accordance with another aspect of the present invention, a method is practiced in the distributed system having repositories for storing packages. A selected package is provided on multiple repositories. Response times are determined from the repositories in which the selected package is provided. The selected package is then retrieved from the repository that has the shortest response time. As test communication may be used to assist in determining response times.

In accordance with yet another aspect of the present invention, a method is practiced by a client system within a distributed system. The distributed system includes a publishing master that has an index of packages available for installation. At least a portion of the index is requested from the publishing master. This portion of the index is received at the client system and processed to identify a selected package to install. Subsequently, a copy of the selected package is obtained and the selected package may then be installed.

In accordance with a further aspect of the present invention, a method is practiced in a distributed system that has a client and a storage holding packages that are available for downloading. Data is obtained that identifies packages that are available for downloading. The packages to which the client subscribes are identifies as are the packages that are already downloaded on the client. Selected packages are then downloaded to the client from the storage. The selected packages are those which are identified for installation, identified as packages to which the client subscribed and identified as not yet being downloaded to the client.

Brief Description of the Drawings

An illustrative embodiment consistent with the principles of the present invention will be described below relative to the following drawings.

FIGURE 1 depicts a block diagram of a distributed system that is suitable for practicing the illustrative embodiment.

FIGURE 2 illustrates an exemplary client computer system in more detail.

FIGURE 3 is a flow chart illustrating the steps that are performed to add a package to the packages that are available for installation.

FIGURE 4 depicts the format of a package index.

FIGURE 5 is a flow chart illustrating the cycle of steps that is performed to keep client computer systems updated.

FIGURE 6 is a flow chart illustrating the steps that are performed to update a single client computer system.

FIGURE 7 is a flow chart illustrating the steps that are performed to identify what packages need to be updated on a given client computer system.

FIGURE 8 illustrates how a list of packages that need to be updated is generated from other data.

FIGURE 9 depicts an example of a portion of a subscription list.

FIGURE 10 illustrates the steps that are performed to update a given client computer system.

FIGURE 11 depicts the logical organization of an exemplary notification.

FIGURE 12 is a flow chart illustrating the steps performed to dynamically determine a repository to use in retrieving a package for installation when the package is stored on multiple repositories.

Detailed Description of the Invention

The illustrative embodiment provides a lightweight global distribution mechanism. The distribution mechanism is global in that all of the client computer systems within a distributed system may be serviced using this mechanism. The mechanism is lightweight in that it requires minimal overhead. No registration is used and no registry is maintained. The distribution mechanism is anonymous in that a client computer system may install packages unbeknownst to a publishing master.

Figure 1 depicts an example of a distributed system 10 that is suitable for practicing the illustrative embodiment. Those skilled in the art will appreciate that the depiction in Figure 1 is intended to be merely illustrative and not limiting of the present invention. The present invention may be practiced in environments that use a different number of repositories, client systems and publishing masters than shown in Figure 1. In addition, different interconnection arrangements may be provided while still practicing the present invention.

The distributed system 10 includes a publishing master 12. The publishing master 12 is responsible for maintaining a package index that serves as an index of packages that are available for installation. This index itemizes the packages that may be installed at the client systems 14. The publishing master 12 may be a dedicated server computer system. Alternatively, the publishing master may be realized as a process that runs on a computer system along with other processes. Those skilled in the art will appreciate that the publishing master may be implemented using a variety of different types of computer systems including personal computers, workstations, mini-computers or mainframe systems.

The client systems 14 are computer systems on which the packages may be installed or downloaded. A "package," as used herein, refers to a logical quantity of information that may be encapsulated and sent to a client system 14 for installation. A package may contain one or more computer programs, patches, documents, data, records, HTML files or the like. In general, a package may contain any information that is useful for distribution to the client systems 14 within the distributed system 10. As used herein, a "patch" is a section of computer program instructions that is provided to update or repair a deficiency in an existing routine or program. The patches may provide additional features or functions to an existing version of a program between releases of versions of the program. The client systems 14 may take many forms. For example, the client systems 14 may be realized as personal computer systems, as server computer systems, as workstations, as network computers, as intelligent digital devices or as other types of devices. The client systems 14 are the entities on which the packages are installed. Thus, if a package contains a patch, the client system 14 receives the patch and installs the patch therein.

The distributed system 10 may include one or more repositories 16. The repositories 16 may be realized as generic storage servers. The packages that are indexed by the package index are stored on the repositories 16. A single package may be stored on multiple repositories 16. As will be explained in more detail below, the client systems 14 may obtain packages from the repositories 16 and download and/or install them.

The distributed system 10 may be a web-based system, such as the Internet, an intranet or an extranet. In such a case, the publishing master 12 may be realized as a web server that provides the client systems 14 access to the publishing index. The repositories 16 support anonymous File Transfer Protocol (FTP) and/or Hypertext Transfer Protocol (HTTP) access to the packages stored therein.

Figure 2 depicts examples of a format for a client system 14. The client system 14 includes a central processing unit (CPU) for overseeing operations within the client system 14. The CPU 20 executes computer program instructions to perform activities. The client system 14 may includes input devices, such as a keyboard 22 and a mouse 24. The client system 14 may also include output devices, such as a video display 26. The client system 14 may include a network adapter 28 for interfacing the client system with a computer network 30. The client system 14 may contain a modem 32 for communicating with remote computing resources. The modem 32 may be realized as a conventional modem, a wireless modem or a cable modem, for example.

The client system includes both primary storage 40 and secondary storage 34. The storages 34 and 40 may include removable computer readable medium, including removable magnetic disks, optical disks and/or magneto-optical disks. The secondary storage 34 may hold a number of lists 36 (which will be described in more detail below) that are used in determining which packages to download and/or install. Secondary storage 34 may also hold a copy of a subscription list 38 that identifies the packages to which the client system subscribes (i.e. those packages for which the client system wants updates).

The primary storage 40 may hold a number of different types of programs and data. The primary storage may include a web browser 41 for enabling the client system 14 to communicate with web servers and, in general, to provide web-based access to remote computing resources. The primary storage 40 holds three modules 44, 46 and 48 that perform respective roles in updating packages on the client system 14. As will be described in more detail below, the check-pkgs module 44 is responsible for identifying what packages need to be updated and installed at the client system. The sync-pkgs module 46 is responsible for performing the updates, and the notification module 48 is responsible for generating notifications that notify the completion of updates.

The client system 14 may run an operating system, such as the Solaris™ operating system from Sun Microsystems, Inc. of Palo Alto, California. The operating system may include a scheduler 50 (e.g. Cron) that is responsible for scheduling activities within the client system 14. As will be described in more detail below, the scheduler may be responsible for triggering updates so that the packages that are installed on the client system 14 may be ensured to be current. The client system may include various transmission media for transmitting signals to perform desired operations.

Those skilled in the art will appreciate that the depiction of a client system 14 shown in Figure 2 is intended to be merely illustrative and not limiting of the present invention. Each of the client systems 14 need not have the same configuration. Moreover, the present invention may be practiced with computer systems that include different components than those depicted in Figure 2.

The global distribution mechanism of the illustrative embodiment facilitates the addition of packages so that the packages are available for installation. Figure 3 depicts a flow chart of the steps that are performed to add a package. Initially, the package is staged on a repository 16 (step 52 in Figure 3). The choice of repository for storing the package may be based upon a number of factors, including available storage capacity, proximity to subscriber client systems 14 and load balancing considerations. For example, a single package may be stored on multiple repositories 16, and a client system may dynamically choose what package to download based on current loads, as will be described in more detail below. The package index stored on the publishing master 12 is then updated to include a listing for the newly staged package (step 54 in Figure 3). Conversely, the distribution mechanism facilitates the removal of packages by removing the packages from the index, and may entail deletion of the package from the repositories 16.

Figure 4 depicts the logical format of the package index 60. The package index includes a number of entries where each entry is associated with a particular package. In Figure 4, entries 70, 72, 74 and 75 are associated with separate packages. Each entry includes a name field 62 that identifies the name of the package (e.g. coyote). Each entry also includes a version field 64 that identifies the version of the package (e.g. 2.0). Each entry additionally includes a type field 66 that identifies the type of package (e.g. "O") and a location field 68 that identifies a location of the package (e.g. http://www.arizona.com/coyote). Multiple locations may be specified (see entry 75) where the package is stored at multiple locations.

The type field 66 may specify whether the package is a required update ("R"), an optional update ("O"), a patch update ("P") or a stealth update ("S"). A required update is a package that is flagged for update on any system that either does not currently have the package installed or that has a version lower than the version indicated in the package index field. An optional update is flagged for update on any system that has subscribed to the package and that does not have the package installed or has a version lower than the version indicated in the package index field. A patch update can either be a required package or an optional package. Required patch packages are by default installed on every system. Optional patch packages are only installed on client systems that subscribe to the patch. The "P" designator in the type field 66 may be followed by a hyphen and an optional package name. In such a case, the patch 11 be updated only if the package with the specified optional package name is currently installed and subscribed to by the client system. A stealth update is a required update to an optional package.

Figure 5 is a flow chart illustrating in the steps that are performed to update packages on a given client system. Initially, the update is triggered (step 80 in Figure 5). The update may be triggered at periodic intervals by the scheduler 50 or may be triggered in a different fashion. For example, the update may be triggered by a user explicitly requesting an update or by other events that indicate that update is required. The update is then performed to install packages as needed (step 82 in Figure 5). In particular, the modules 44, 46 and 48 are executed. The system then waits (step 84 in Figure 5) until an update is again triggered (see step 80 in Figure 5).

Figure 6 is a flow chart illustrating the steps that are performed to update a client system 14. Initially the check-pkgs module 44 is activated to determine what packages need to be updated (step 90 in Figure 6). Figure 7 is a flow chart that illustrates the steps that are performed by the check-pkgs module. Initially, a client system 14 retrieves the package index 60 from the publishing master 12 and generates a list of the published state of the packages (step 100 in Figure 7). Thus, as shown in Figure 8, the package index 110 is retrieved by the client system 14 and processed to generate a publication list 116. The resulting process list reflects the published state of all packages (i.e. a list of packages and versions). The client system then initiates a query to generate a list that identifies the state of all packages installed on the client system 14, including version information for these packages. As shown in Figure 8, the client system 112 generates an installed list 118 that identifies the state of the packages installed on the system.

The client system examines the local subscription list 114 (Figure 8) to generate a list of subscribed packages 120 (step 104 in Figure 7). Figure 8 depicts the format of the local subscription list 114. The local subscription list 114 is a serial listing of the packages to which the local system subscribes.

Those skilled in the art will appreciate that the subscription state list 120 may have a number of different formats and need not be organized as a list per se but may be organized as a set of records, a table or the like. Moreover, the lists may be combined into a single list or in different combinations of lists.

The lists 116, 118 and 120 are then processed to generate a final list 122 that identifies the packages that need to be installed to complete the update (step 106 in Figure 7). Any required updates which are either not installed or have published versions higher than the currently installed packages will be included in the final list 122. Any optional packages which are either not installed or have published versions higher than the currently installed versions and exist in the local subscription list will also be included on the final list 122.

Once the check-pkgs module has completed generation of the final list 122 (see step 90 in Figure 6). The sync-pkgs module is activated (step 92 in Figure 6). Figure 10 provides a flow chart of the steps performed by this module 46. The module 46 obtains the next package name from the final list (step 120 in Figure 10) and installs the name package from the repository on which the package is located. As was mentioned above, the package index contains location information. The package may be installed using FTP or HTTP (step 122 in Figure 10). This process continues in this fashion until the list is fully processed (see step 124 in Figure 10). Thus, the packages are located on the repositories, downloaded to the client systems and installed.

Notifications are generated (step 94 in Figure 6) to notify appropriate personnel via a desired channel (e.g. via email) that an update has happened. Figure 11 depicts a format of a suitable format for a notification 130. A notification should identify the package name 132 that was installed at the client system 14. The notification 130 should also identify the repository server 144 from which the update was obtained and the notification 130 should include a status field 136 that indicates whether there were any errors or not.

Each of the client systems 14 may perform the update process separately. Alternatively, the client systems 14 may be triggered to perform updates by a master mechanism such that all client systems are updated at the same time.

The illustrative embodiment also facilitates dynamic load balancing. In particular, the illustrative embodiment may store a package on multiple repositories and download a copy of the package from one of the repositories based upon current load conditions. Figure 12 is a flow chart illustrating the steps that are performed to realize such dynamic load balancing in the illustrative embodiment. Initially, the client system 14 accesses the package index 60 and determines that there are multiple location fields 68 (step 140 in Figure 12). These locations are on separate repositories 16. The client system then takes steps to determine the respective loads on the repositories where the package is stored. Specifically, the client sends out a test package to each identified repository (step 142 in Figure 12). The test package received by the respective repositories and returned. The responses are received at the client system for the repositories and based upon the time that has elapsed between when the test packet was sent out to the respective repositories and when the respective responses were received, the client system 14 determines response times for the respective repositories (step 144 in Figure 12). The system retrieves the package from the repository with the shortest response time (step 146 in Figure 12). It is presumed that the repository with the shortest response time is the least loaded and best able to fulfill the request.

The dynamic load balancing provided in the illustrative embodiment allows the strategic location repositories across the network. There is no need for the repository to be crafted for one particular client. The client chooses the repository that is best suited for the client needs. In addition, the scheme allows a server to go down or be services without fear of endangering the client systems 14.

Those skilled in the are will appreciate that the present invention may be practiced with implementations that use a different programmatic structure. The respective phases of the updates need not be performed by separate modules but rather may be performed by a single integrated program or by multiple disparate programs.

While the present invention has been described with reference to an illustrative embodiment thereof, those skilled in the art will appreciate that various changes in form and detail may be made without departing from the intended scope of the present invention as defined in the appended claims.